CINET¹ GDS-calculator
Graph Dynamical Systems Visualization

Group CINETgds
Sichao Wu, Yao Zhang
Virginia Tech
Dec. 6 2012

CS5604 Final Project Presentation

¹ CINET is supported by the NSF under Grant No. 1032677
Recap: What is GDS?

- **GDS**: Graph Dynamic Systems.
- Five elements:
  - Graph.
  - Vertices, which represent states.
  - Edges, which indicate interacting vertices.
  - Vertex functions, which quantify vertex state changes.
  - Update scheme: synchronous, sequential, block, etc.
Motivation

- System behaviors are of practical interest.
  - Examples
    - How epidemics spread.
    - How information spreads on Twitter.

- Understanding of system dynamics provides insights on how to control them.
  - Examples
    - How to minimize the spread of epidemics.
    - How to increase the probability that a marketing campaign goes viral.
    - How to encourage healthy youth behavior (e.g., avoid smoking, excessive drinking).

Sociology and computer science community call these diffusing entities contagions.
We are recasting the real problem in terms of a GDS
Have the model, then what?

- We are interested in the long term dynamical properties of the GDS system.
- **Phase space** reflects the long term dynamics.
  - Vertex state: For vertex $i$, we sign a state $x_i$ from the state space, such as $\{0, 1\}$.
  - System state: states for all vertices, we write $x = \{x_1, x_2, \ldots, x_n\}$.
  - Phase space describes the system state transitions.
Phase space example

• Inputs
  – **Graph**: Circle
    ![Circle Graph]
  – Vertex state = \{0,1\}
  – Vertex functions: \text{nor}_3
  – Update scheme:
    • Sequential with order \(1,2,3,4\)
    • Synchronous

Phase space: sequential update

Phase space: synchronous update
Character of phase space

- Functional equivalence (FE)
  - What GDS maps produce the same transitions for all system states?
- Dynamic equivalence (DE)
  - What GDS maps produce isomorphomic sets of state transitions?
- Cycle equivalence (CE)
  - What GDS maps produce the same limit cycles?

\[ FE \Rightarrow DE \Rightarrow CE \]
Phase Space Analysis

begin

Read in inputs.

Generate next permutation.

Compute transitions

Determine functional equivalence.

If GDS map J is functionally equivalent to an existing map H, put J into H's equiv. class.

Is another permutation?

yes

no

Print to output the functionally equivalent classes (FEC).

For each FEC, print the limit cycles.

From FEC, determine cycle equivalent classes (CEC).

For each CEC, print cycle summary with multiplicities.

end
Task - Overview

- GDS calculator visualization system has three main tasks:
  - Visualizing bar charts.
  - Visualizing phase space transition graphs.
  - Exporting to graphs and charts to files.
Task – Visualizing bar chars

- Bar Charts we have implemented:
  - the number of cycles for each cycle equivalent set.
  - the number of permutations for each permutation equivalent set.
  - the number of cycles for each permutation.
  - the maximum cycle length for each permutation.
  - the number of permutations for a specific cycle length.
Task – Visualizing graphs

- **Graph we have implemented:**
  - functional state transitions.
  - cycle state transitions.

- **Exporting to graphs and charts to files**
  - Export bar charts: png, pdf.
  - Export graphs: png, ge.
Design – MVC Architecture

- **View**: visualization part for GDS-VIS System
  - visualizing bar charts.
  - visualizing state transition graphs.

- **Controller**: manipulating data in the Model
  - parsing configuration file to generate data.
  - managing data.
    - summarizing data to generate bar charts.
    - collecting data to generate graphs.
  - exporting graph to files.
Design - Model

- Model: data used in GDS-VIS system
  - Permutation
  - Node state
  - Functional transition
  - Cycle transition
  - Functional equivalence
  - Cycle equivalence
Implementation - Environment

- Implementation environment:
  - Java version: JDK 1.7
  - IDE: Eclipse Juno

- Tools:
  - jdom: parsing XML files
  - jfreechart: visualizing bar charts
  - gephi-toolkit: visualizing state transformation
  - itext: exporting to pdf files
Implementation – Class diagram

- **Model**

  - **Permutation**
    - name: String
    - sequence: int [ ]
    + getName() : String
    + getSequence() : int[]
    + Permutation(String, int, LinkedList<Integer>)

  - **NodeState**
    - name: String
    - sequence: int [ ]
    + getName() : String
    + getSequence() : int[]
    + NodeState(String, int, LinkedList<Integer>)

  - **FuncTransition**
    - edge_list: LinkedList<Edge>
    - name: String
    + addEdge(Edge) : void
    + FuncTransition(String)
    + getEdgeList() : LinkedList<Edge>
    + getName() : String

  - **Cycle**
    - edge_list: LinkedList<Edge>
    - num: int
    + addEdge(Edge) : void
    + Cycle()
    + getEdgeList() : LinkedList<Edge>
    + getNum() : int

  - **CycEquivalence**
    - cyc_equi_set: Set<String>
    - name: String
    - num: int
    + addPermutation(String) : void
    + CycEquivalence(String)
    + getCycEquiSet() : Set<String>
    + getName() : String
    + GetPNum() : int

  - **FuncEquivalence**
    - func_equi_set: Set<String>
    - name: String
    + addPermutation(String) : void
    + FuncEquivalence(String)
    + getFuncEquiSet() : Set<String>
    + getName() : String

  - **Edge**
    - from: String
    - label: String
    - to: String
    + Edge(String, String)
    + Edge(String, String, String)
    + getFrom() : String
    + getLabel() : String
    + getTo() : String

  - **CycTransition**
    - cyclist: LinkedList<Cycle>
    - name: String
    - num: int
    + addCycle(Cycle) : void
    + CycTransition(String)
    + getCycList() : LinkedList<Cycle>
    + getName() : String
    + maxCycleNum() : int
Controller and View

```java
class ModelManager
{
    private ModelManager manager = null;
    private ParseGraph pg = null;

    public ModelManager(ParseGraph parseGraph)
    {
        this.manager = parseGraph;
    }

    public ParseGraph getParseGraph()
    {
        return this.manager;
    }

    public void display()
    {
        this.manager.display();
    }

    public void getInstance(ParseGraph parseGraph)
    {
        this.manager.getInstance(parseGraph);
    }

    public int getMaxCycLength()
    {
        return this.manager.getMaxCycLength();
    }

    public int getTotalNumberOfPIForCycle()
    {
        return this.manager.getTotalNumberOfPIForCycle();
    }
}
```

```java
class ParseGraph
{
    private int numOfNodes;
    private Element root;

    public ParseGraph()
    {
    }

    public Element getRoot()
    {
        return this.root;
    }

    public int getNumOfNodes()
    {
        return this.numOfNodes;
    }
}
```
Implementation – Configuration file

```xml
<?xml version="1.0"?>
<GDS>
  <graph>
    <numnodes>4</numnodes>
  </graph>

  <permutations>
    <permutation name="PA">
      <nodeid>0</nodeid>
      <nodeid>1</nodeid>
      <nodeid>2</nodeid>
      <nodeid>3</nodeid>
    </permutation>
  </permutations>

  <nodeinitialstate>
    <nodeid>0</nodeid>
    <nodeid>1</nodeid>
    <nodeid>2</nodeid>
    <nodeid>3</nodeid>
  </nodeinitialstate>

  <transition permutation="PA">
    <cycle>
      <edge>
        <from>SA</from>
        <to>SB</to>
        <label></label>
      </edge>
      <edge>
        <from>SB</from>
        <to>SA</to>
        <label></label>
      </edge>
    </cycle>
    <cycle>
      <edge>
        <from>SA</from>
        <to>SA</to>
        <label></label>
      </edge>
    </cycle>
  </transition>
</GDS>
```
Implementation - GUI

Number of Cycles for Each Permutation
Future work

- Improving GUI implementation.
- Visualizing more statistical charts.
- Integrating with the CINET project.
Thanks